School of Computation, Information and Technology
Technische Universität München

Lehrstuhl für
Rechnerarchitektur & Parallele Systeme
Prof. Dr. Martin Schulz
Dominic Prinz
Jakob Schäffeler

Lehrstuhl für
Design Automation
Prof. Dr.-Ing. Robert Wille
Stefan Engels

Einführung in die Rechnerarchitektur

Wintersemester 2024/2025

Übungsblatt 11: Pipelining 13.01.2025 – 17.01.2025

1 Pipelining Speedup

Element	Parameter	Delay (ps)
Register read	t_{RegRead}	40
Register setup	t_{RegSetup}	50
Multiplexer	t_{mux}	30
AND-OR gate	$t_{\text{AND-OR}}$	20
ALU	t_{ALU}	120
Decoder (Control Unit)	t_{dec}	25
Extend unit	t_{ext}	35
Memory read	t_{MemRead}	200
Register file read	t_{RFRead}	100
Register file setup	t_{RFSetup}	60

Tabelle 1: Propagation Delays

Nimm an, dass die Komponenten des RISC-V Prozessors die in Tabelle 1 angegebenen Verzögerungen haben (eine Verzögerung gilt zwischen jedem Eingang und Ausgang der Komponente). Für nicht gelistete Komponenten soll eine Verzögerung von 0 ps angenommen werden.

a) Zeichne die *kritischen Pfade* aller Pipelining-Stufen des RISC-V-Prozessors in Abbildung 1 ein und bestimme deren Länge (in ps).

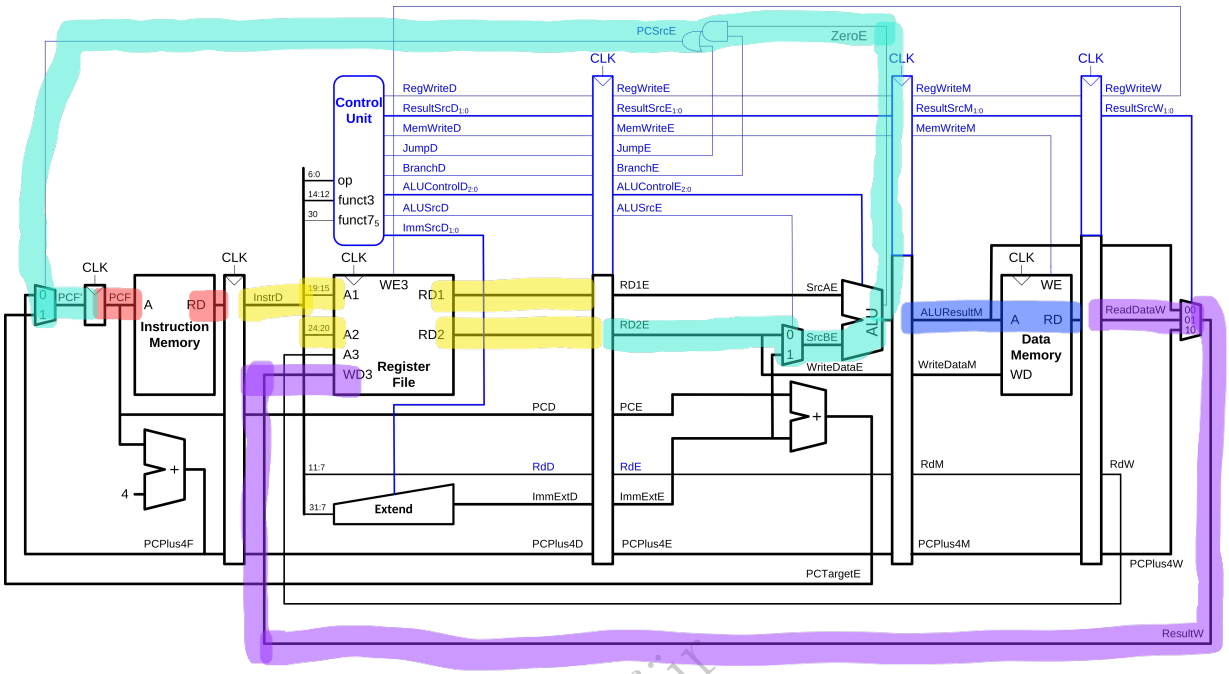
Fetch: $t_{\text{RegRead}} + t_{\text{mux}} + t_{\text{RegSetup}}$: 290 ps

Decode: $t_{\text{RegRead}} + t_{\text{RFRead}} + t_{\text{RegSetup}}$: 190 ps

Execute: $t_{\text{RegRead}} + t_{\text{mux}} + t_{\text{ALU}} + t_{\text{AND-OR}} + t_{\text{mux}} + t_{\text{RegSetup}}$: 290 ps

Memory: $t_{\text{RegRead}} + t_{\text{mux}} + t_{\text{RegSetup}}$: 290 ps

Write Back: $t_{\text{RegRead}} + t_{\text{mux}} + t_{\text{RegFileSetup}}$: 130 ps



b) Bestimme die minimale Länge des Taktzyklus für den RISC-V Prozessors mit fünfstufiger Pipeline (Abbildung 1). Welcher Speed-up kann im Vergleich zum Prozessor ohne Pipeline (Abbildung 3, vgl. kombinatorischer Prozessor vom letzten Übungszettel) theoretisch erreicht werden? Warum ist dieser Speed-up nicht gleich der Anzahl der Pipeline-Stufen?

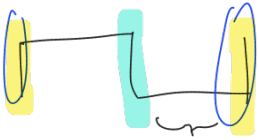
min. Taktlänge: 290 ps (max. aus Delays der Pipelinestufen)

Taktlänge$_{\text{Single Cycle}}$: 750 ps

Speedup: $\frac{750\ ps}{290\ ps} = 2,59$

Speedup ist nicht 5 da:

- Overhead durch Zwischen-FF
- Alle Stufen brauchen nicht gleich lang



c) Wie ändert sich die Taktlänge wenn die Registerbank bei fallender Flanke geschrieben wird und die Registerbank somit erst in der zweiten Hälfte des Taktes gelesen werden kann?

Decode muss $\frac{1}{2}$ Taktlänge auf Writeback warten

$t_{RegRead}$ $t_{RegFileRead} + t_{RegSetup} = 150\,ps$

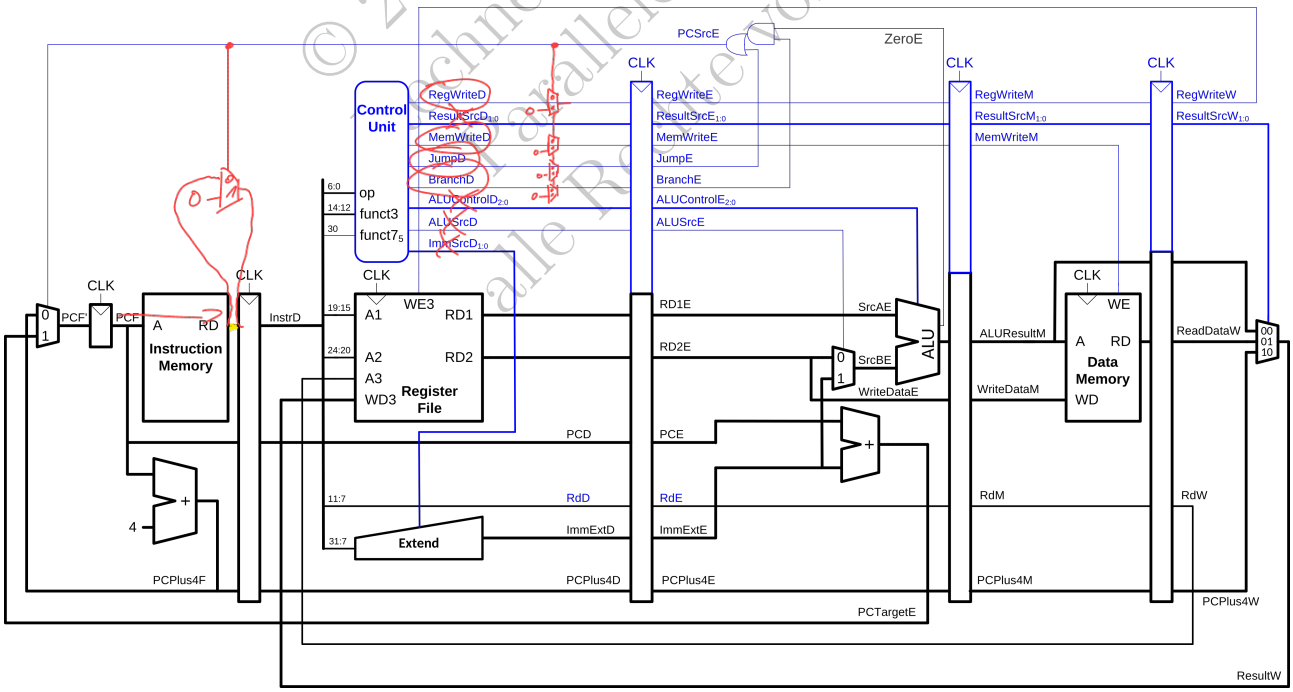
$\frac{1}{2}$ $2 \cdot 150\,ps = 300\,ps$

⟶ neue Taktlänge im Prozessor: 300 ps

2 Flushen der Pipeline (ohne Hazard Unit)

Aus der Vorlesung wissen wir, dass eine Stage der Pipeline geflushed werden kann indem man rücksetzbare Register zwischen Pipelinestages verwendet. In dieser Übung wollen wir Flushing umsetzen *ohne* die Register selbst zu verändern.

Erweitere dazu den RISC-V Prozessor in Abbildung 1, sodass das Flushen der Pipeline bei den Befehlen `beq` und `jal` unterstützt wird. Füge dazu möglichst wenig Logik zum Datenpfad des Prozessors hinzu. Es darf angenommen werden, dass (zusätzlich zu `addi zero, zero, 0`) eine Instruktion die nur aus Nullen besteht, einem NOP entspricht. Bestehende Komponenten dürfen nicht verändert werden.



Setzt diese Änderung in Digital um und verifiziere das Verhalten mit dem beigefügten Testprogramm.

Nach Sprung müssen immer 2 Befehle geflusht werden (Fetch, Decode)

3 Pipelining mit Hazard Unit

Der RISC-V Prozessor mit Pipelining und Hazard Unit aus Abbildung 2 führt den folgenden Programmcode aus.

```
s1    addi t1, zero, 52
s2    addi t0, t1, -4
s3    lw t3, 16(t0)
s4    sw t3, 20(t0)
s5    xor t2, t0, t3
s6    or t2, t2, t3
```

Zeile Register

RAW S2, S1, +1 Forwarding Memory → Exec

RAW S4, S3, +3 NOP + Forwarding WB → Exec

RAW S3, S2, +0 Forwarding Memory → Exec

RAW S6, S5, +2 Forwarding Mem → Exec

a) Erläre welche Konflikte in dem Programm auftreten. Welche davon können durch Forwarding gelöst werden und von welcher Pipeline-Stufe muss geforwarded werden? Welche Konflikte müssen durch Stalling gelöst werden?

b) Wie viele Taktzyklen sind erforderlich um alle Befehle in die Pipeline zu laden?

Cycle	F	D	E	M	WB
1	addi				
2	addi	addi			
3	lw	addi	addi		
4	NOP	lw	addi	addi	
5	sw	NOP	lw	addi	addi
6	xor	sw	NOP	lw	addi
7	OR	xor	sw	NOP	lw

→ 7 Zyklen

c) Mit den Delays aus Tabelle 1 hat der Prozessor aus Abbildung 2 eine Taktlänge von

350ps.

Auf dem Prozessor wird ein Programm ausgeführt das aus 25% `lw`, 10% `sw`, 11% `beq`, 2%`jal` und 52 % R- oder I-Typ ALU Instruktionen besteht. Nimm an, dass 40% der `lw` von Befehlen gefolgt werden die das Ergebnis direkt verwenden und 50% der Branches genommen werden. Bei einer falschen Sprungvorhersage müssen 2 Befehle geflushed werden.

Was ist die vorraussichtliche Laufzeit wenn das Programm aus 10^{11} Instruktionen besteht? Das initiale Laden der Pipeline kann ignoriert werden.

lw : $0{,}4 \cdot 2 + 0{,}6 \cdot 1 = 1{,}4$

beq: $0{,}9 \cdot 3 + 0{,}5 \cdot 1 = 2$

jal : 3 cycles/instr

$0{,}25 \cdot 1{,}4 + 0{,}1 \cdot 1 + 0{,}11 \cdot 2 + 0{,}52 \cdot 1 + 0{,}02 \cdot 3 = 1{,}25$

$1{,}25 \cdot 10^{11}\ \text{Instr} \cdot 350\ \text{ps} = 44\text{s}$

SingleCycle: 79s
MultiCycle: 155s

4 Datenabhängigkeiten und Pipeline-Konflikte (Hausaufgabe)

Bearbeitung und Abgabe der Hausaufgabe 11 auf `https://artemis.in.tum.de/courses/401` bis **Sonntag, den 19.01.2025, 23:59 Uhr**.

Ziel dieser Übung ist es, ein vorgegebenes Programm durch Einfügen von NOPs und Umordnen der Befehle zu beschleunigen. Betrachte das folgende RISC-V Programm und nimm an, dass es auf einem RISC-V Prozessor mit fünfstufiger Pipeline *ohne Hazard Unit* (siehe Abbildung 1) ausgeführt werden soll. Beachte, dass das Ergebnis des `lw` Befehls erst am Ende von Writeback in die Registerbank des Prozessors in Abbildung 1 geschrieben wird.

```
i1:     and  t0, t1, t2
i2:     addi t2, t3, -13
i3:     xor  t4, t2, t1
i4:     beq  t0, x0, else
i5:     addi t2, t5, 17
i6:     lw   t0, 64(t3)
i7:     or   t3, t0, t5
i8:     j end
    else:
i9:     sw   t3, 0(t0)
i10:    xor  t0, t3, t4
i11:    and  t5, t2, t4
i12:    add  t4, t1, t0
    end:
```

```
i13:    nor  t2, t3, t0
i14:    sub  t6, t6, t1
```

a) Gib alle Datenabhängigkeiten zwischen den Befehlen des Programms an (RAW, WAR und WAW).

b) Finde alle auftretenden Pipeline-Konflikte (Daten- und Kontroll-Konflikte). Gib jeweils die involvierten Befehle an und erkläre warum es zu einem Konflikt kommt.

Hinweis: Ob ein Konflikt vorliegt oder nicht kann von dem `beq`-Befehl in Zeile 4 abhängen.

c) Behebe alle Pipeline-Konflikte durch Einfügen der minimalen Anzahl an NOP-Befehlen.

d) Minimiere die Anzahl der NOP-Befehle durch Umordnen der Befehle. Nimm dabei an, dass der Prozessor auch das Löschen (flushen) von Pipeline-Registern unterstützt. Die Semantik des Programms soll dabei nicht verändert werden. Das heißt, dass das veränderte Programm sowohl auf dem Prozessor mit Pipelining als auch auf dem Single-Cycle Prozessor die Datenabhängigkeiten des ursprünglichen Programms bewahren soll und weiterhin dieselben Instruktionen ausführt (aber ggf. in anderer Reihenfolge).

An den Befehlen selbst darf nichts verändert werden. Lediglich die Reihenfolge darf vertauscht werden. NOPs werden im Assembly durch Zeilen dargestellt in denen einfach `NOP` steht.

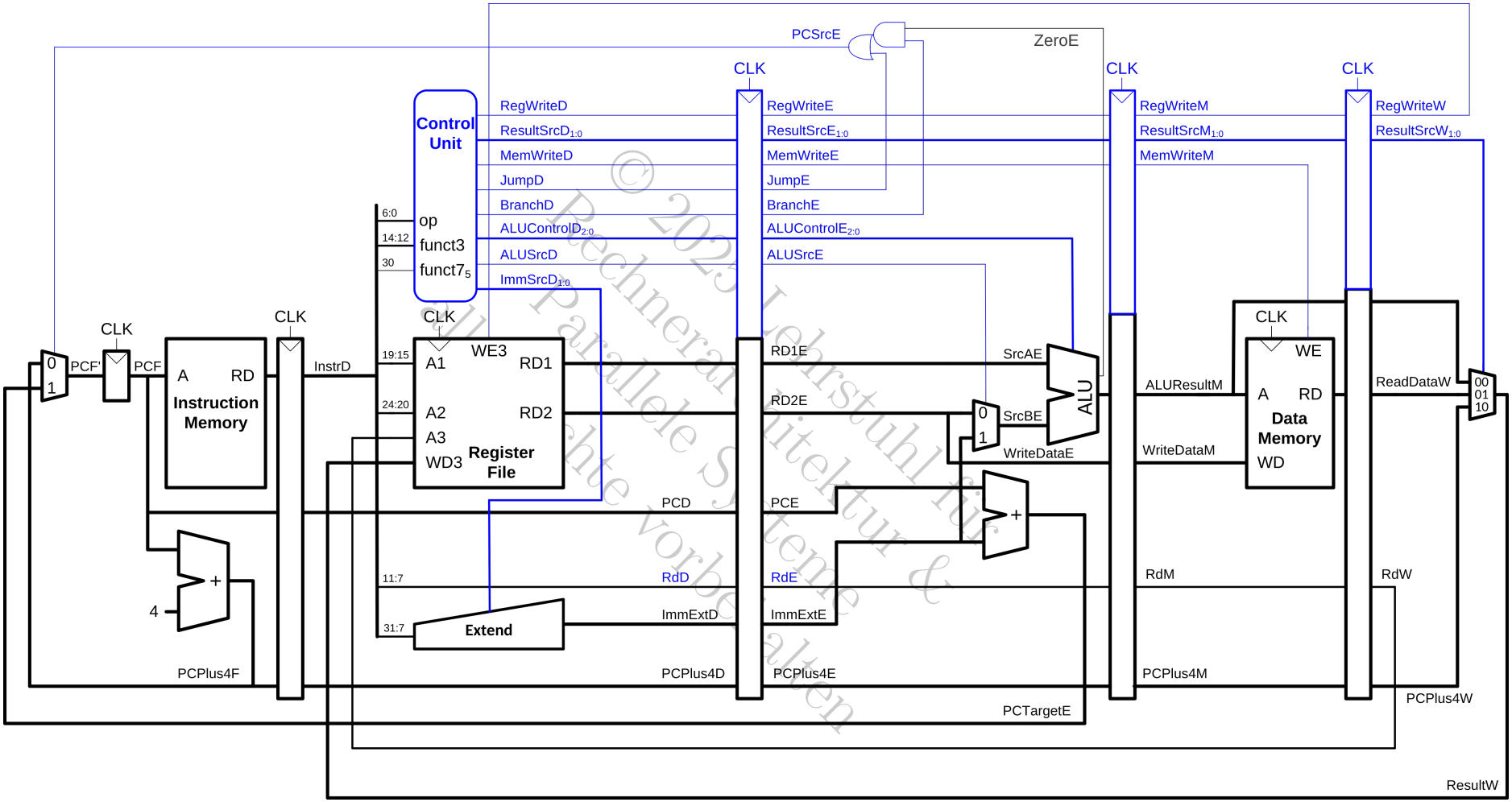


Abbildung 1: Schaltbild des Pipelined RISC-V Prozessors one Hazard Unit

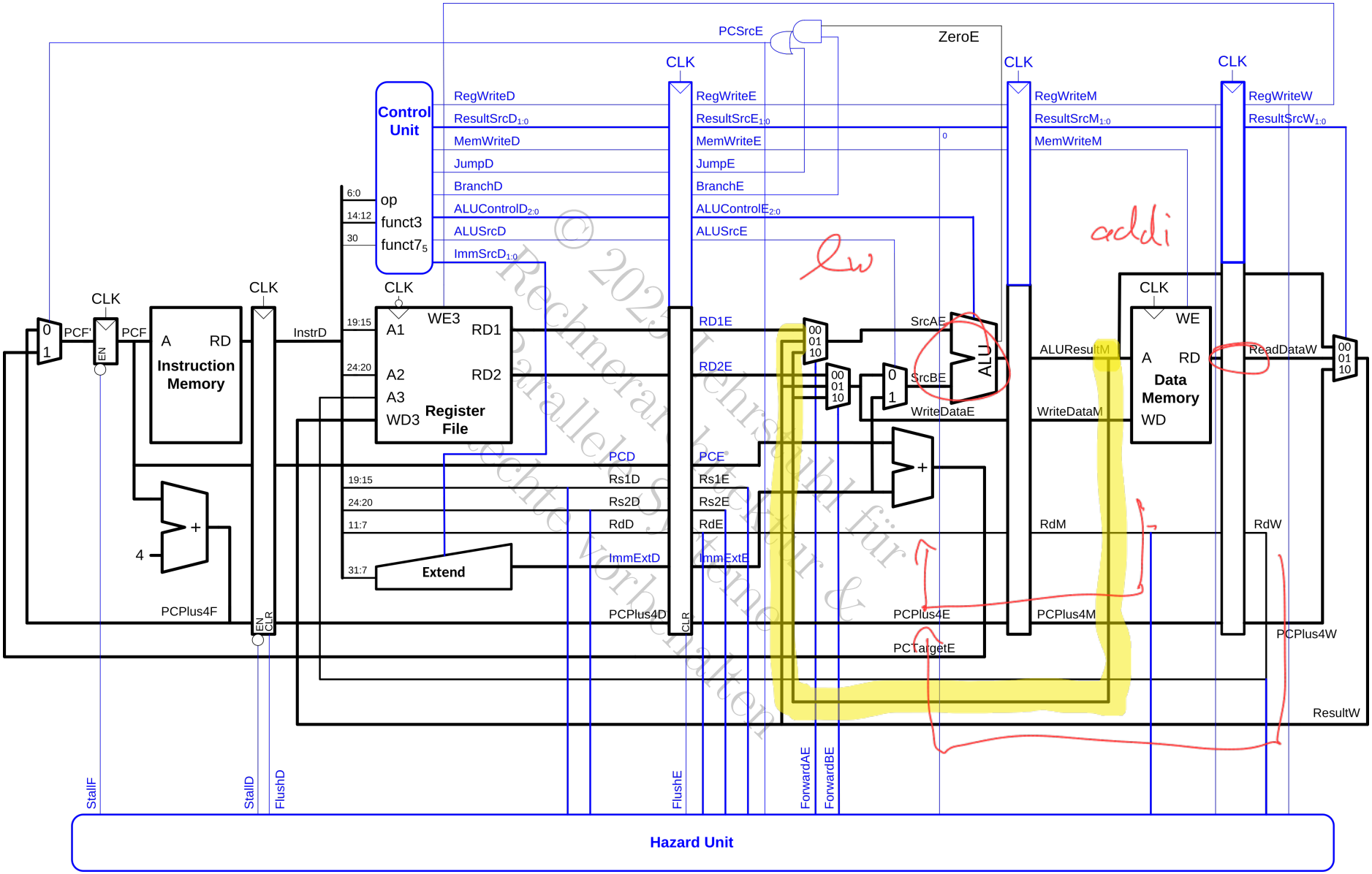


Abbildung 2: Schaltbild des Pipelined RISC-V Prozessors mit Hazard Unit

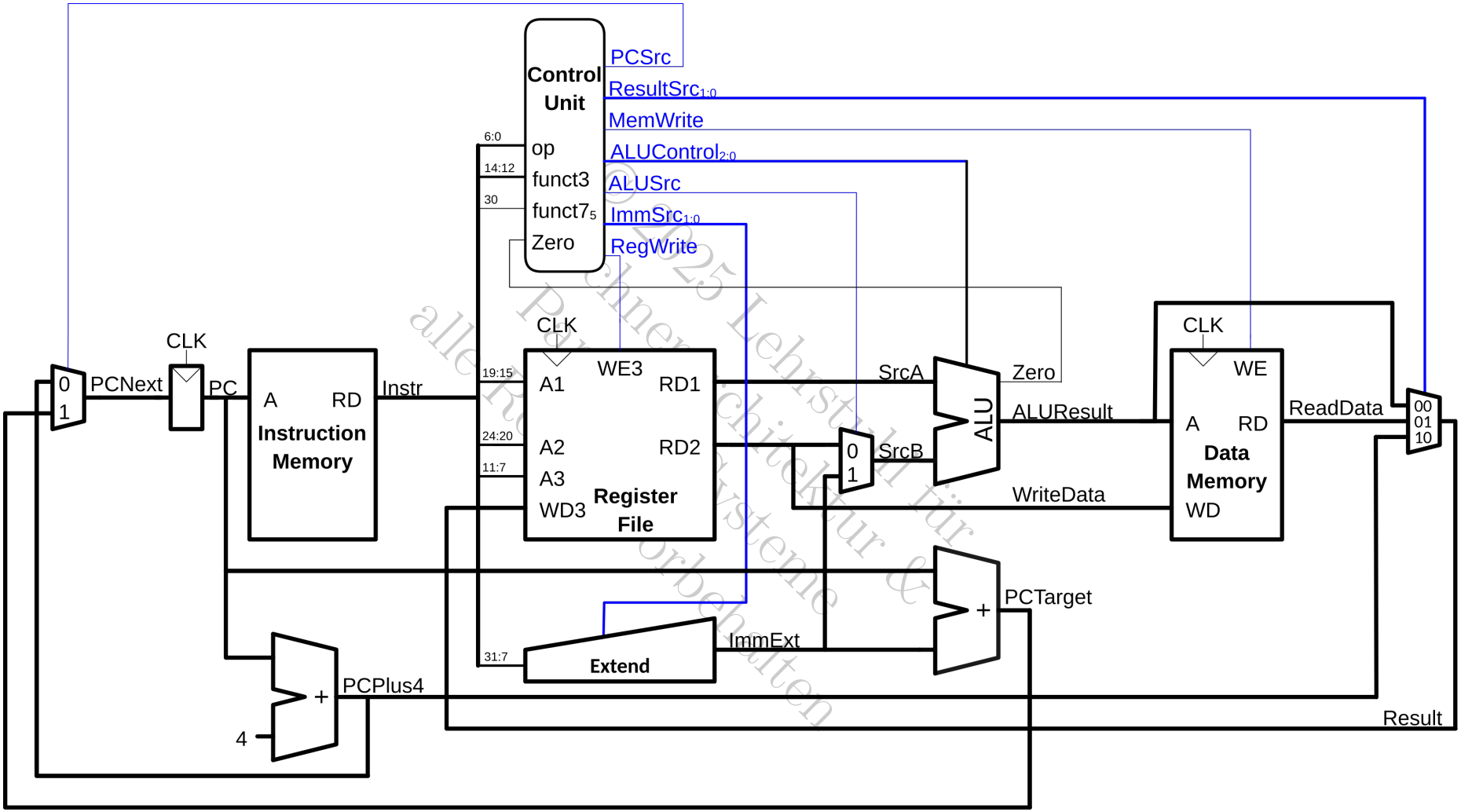


Abbildung 3: Schaltbild des Single-Cycle RISC-V Prozessors